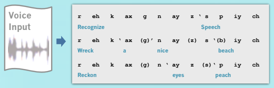
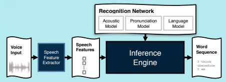
Efficient Automatic Speech Recognition on the GPU

Jike Chong, Ekaterina Gonina, Kurt Keutzer
Department of Electrical Engineering and Computer Science, University of California, Berkeley
jike@eecs.berkeley.edu, egonina@eecs.berkeley.edu, keutzer@eecs.berkeley.edu

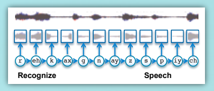
Parallel Computing Lab

Automatic Speech Recognition (ASR)



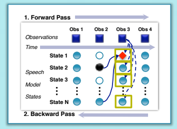


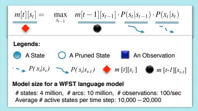
- Automatic speech recognition (ASR) allows multimedia content to be transcribed from acoustic waveforms to word sequences
- This is a challenging task as there can be exponentially many ways to interpret an utterance (a sequence of phones) into words
- ASR uses the hidden Markov model (HMM)
  - States are *hidden* because phones are *indirectly observed* through the waveform
  - Must infer the *most likely interpretation* while taking the language model into account

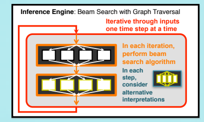


- An ASR application extracts features from a waveform, compares them to the recognition network, and infers the most likely word sequence
- The recognition network is compiled off-line from a variety of knowledge sources and trained using powerful statistical learning techniques
- The inference process traverses a graph-based recognition network using the Viterbi algorithm
- This architecture is modular and flexible:
  - It can be adapted to recognize different languages by swapping in different recognition networks and different speech feature extractors, the inference engine would remain unchanged

ASR Characteristics and Software Architecture



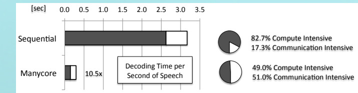




- The Viterbi algorithm is used to infer the most likely interpretation of the observed waveform
  - It has a forward pass and a backward pass
- The forward pass has two phases of execution
  - Phase 1 evaluates the observation probability, which matches the observation to known speech model states (dashed arrows)
  - Phase 2 references historic information and evaluates the likelihood of going from one time step to the next (solid arrows)
- The fine-grained concurrency in ASR lies in the evaluation of each algorithmic step in Phase 1 and Phase 2:
  - The algorithm typically tracks 10,000 – 20,000 alternative interpretations (or active states) at the same time
- The various components of the recognition network can be composed using Weighted Finite State Transducer (WFST) techniques
  - Models typically contain millions of states and tens of millions of arcs
- The software architecture of the inference process is defined above:
  - There is an iterative outer loop that examines one input observation (corresponding to a 10ms time step) at a time
  - The two phases of execution dominate each iteration
  - Concurrency lies in each algorithm steps in the two phases
- The software architecture presents significant challenges when implemented on the GPU, see below for details

Results

- The speech model is taken from the SRI CALO real time meeting recognition system
- The acoustic model includes 52K triphone states clustered into 2,613 mixtures of 128 Gaussian components
- The pronunciation model contains 59K words with 80k pronunciations
- A small back-off bigram language model with 167k bigram transitions was used
- Results presented are based on:
  - Manycore: GTX280 GPU, 1.296GHz, 1GB
  - Sequential: Core i7 920, 2.66GHz, 6GB
- The accuracy achieved for CPU and GPU implementations are identical



- An order of magnitude speed up was achieved as compared to a SIMD optimized sequential implementation running on one core of Core i7 processor
- The compute intensive phase was accelerated by 17.7x
- The communication intensive phase was accelerated by 3.7x
  - Synchronization overhead is dominating execution time as we leverage more computing platform parallelism

Key Lessons

- Challenge 1:
  - Having the freedom to improve the data layout of the runtime data structures is crucial to effectively exploit the fine-grained concurrency in ASR
- Challenge 2:
  - An effective sequential algorithm often cannot be directly translated into a parallel algorithm, e.g. Memoization does not have an equivalent efficient parallel form, the sort-and-filter approach for finding unique element in a list has to be dramatically modified to execute efficiently on a GPU
- Challenge 3:
  - Hardware atomic operation support is extremely important for highly parallel application development
  - Various flavors of atomic instructions with arithmetic and logic operations enable highly efficient implementations for statistical inference problems in machine learning based applications
- Challenge 4:
  - Local synchronization scope important to leverage for relieving global synchronization bottlenecks

Want to learn more about this topic?

Session 2046 - Efficient Automatic Speech Recognition on the GPU

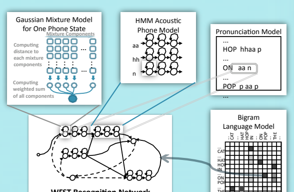
Thursday, September, 23rd, 15:00 - 15:50

Challenge 1:
Handling irregular data structures with data-parallel operations

- In the forward pass of the Viterbi algorithm, there are 1,000s to 10,000s of concurrent tasks that represent the most likely alternative interpretations of the input being tracked
- To track these alternative interpretations, one has to reference a selected subset of data from the WFST Recognition Network with a sparse irregular graph structure
- The concurrent access of irregular data structure requires "uncoalesced" memory accesses in the middle of important algorithm steps, which degrades performance

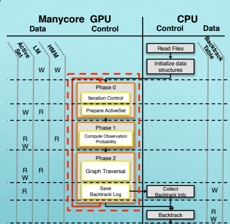
Figure illustrates the various components of a speech model that are compiled together off-line with Weighted Finite State Transducer (WFST) techniques to form a flat probabilistic finite state machine.

The network often contains millions of states and tens of millions of arcs. At runtime, only a small (<1%) subset of the recognition network is referenced at any one time step.



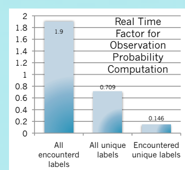
Solution 1:
Construct efficient dynamic vector data structure to handle irregular data accesses

- Instantiate a *Phase 0* in the implementation to gather all operands necessary for the current time step of the algorithm
- Caching them in a memory-coalesced runtime data structure allows any uncoalesced accesses to happen only once for the full time step



Challenge 2:
Eliminating redundant work when threads are computing results for an unpredictable subset of the problems based on input

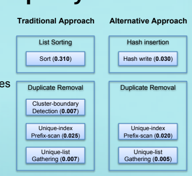
- In a recognition network, there are millions of states, each labeled with one of ~3,000 tied triphone labels
- In a typical recognition sequence, only 20% of the triphone states are used for observation probability computation
- During traversal many of the states being tracked have duplicate labels
- In a sequential execution, memoization is often used to avoid redundant computation
- What do we do for data-parallel platforms?



Real Time Factor shows the number of seconds required to process one second of input data

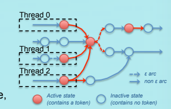
Solution 2:
Implement efficient find-unique function by leveraging the GPU global memory write-conflict-resolution policy

- The traditional approach for finding unique elements in a list involves
  - Sorting the list
  - Detecting consecutive identical elements
  - Scanning to identify unique element indices
  - Copying out unique elements
- Sorting is an computationally expensive step on highly parallel platforms
- For scenarios where the number of possible labels is within 100,000, one can create a hash table of all possible labels
- We leverage the semantics of conflicting non-atomic write of the GPU to use the hash table as a flag array:
  - CUDA guarantees at least one conflicting write to a device memory location to be successful, which is enough to build a flag array
- The alternative "Hash Insertion" step greatly simplifies the find-unique operation



Challenge 3:
Conflict-free reduction in graph traversal to implement the Viterbi beam-search algorithm

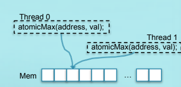
- During graph traversal, active states are being processed by parallel threads on different cores
- Write-conflicts frequently arise when threads are trying to update the same destination states
- To further complicate things, in statistical inference, we would like to only keep the most likely result
- Efficiently resolving these write conflicts while keeping just the most likely result for each state is essential for achieving good performance



A section of a Weighted Finite State Transducer Network

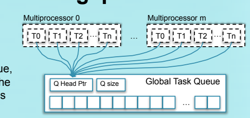
Solution 3:
Implement lock-free accesses of a shared map leveraging advanced GPU atomic operations to enable conflict-free reductions

- CUDA offers atomic operations with various flavors of arithmetic operations
- The "atomicMax" operation is ideal for statistical inference on GPU



- By using it in all threads, the final result in each atomically accessed memory location will be the maximum of all results that was attempted to be written to that memory location
- This type of access is lock-free from the software perspective, as the write-conflict resolution is performed by hardware
- Atomically writing results in to a memory location is a process of reduction
- Hence, this process is performing a ***conflict-free reduction***

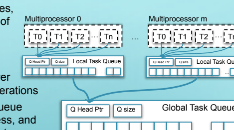
Challenge 4:
Parallel construction of a shared queue while avoiding sequential bottlenecks when atomically accessing queue control variables



- When many threads are trying to insert tasks into a global task queue, significant serialization occurs at the point of the queue control variables

Solution 4:
Use of hybrid local/global atomic operations and local buffers for the construction of a shared global queue to avoid sequential bottlenecks in accessing global queue control variables

- By using hybrid global/local queues, we can eliminate the single point of serialization
- Each multiprocessor can build up its local queue using local atomic operations, which have much lower latency than the global atomic operations
- The writes to the shared global queue are performed in one batch process, and thus are significantly more efficient



This research is supported in part by an Intel Ph.D. Fellowship. This research is also supported in part by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227).